JACOBS UNIVERSITY

Principles of Statistical Modeling (MECS001-340101)

Spring 2022

Data analysis for Seoul bike-sharing service

(Project Report)

Author: *Ali Afsharian* Instructor: *Prof. Dr. Stefan Kettemann*

Abstract

Today, rental bikes are offered in a wide range of urban cities to enhance comfortableness and environmental sustainability. Availability and accessibility of rental bikes to the public are two important aspects of a successful bike-sharing scheme. In order to have a stable supply of rental bikes and reduce waiting time, it is essential to make a prediction for future rental bike demand. This project first discusses an exploratory data analysis for hourly rental bike demand in Seoul based on different predictors such as weather information and time and explores the distribution characteristic of temperature and wind speed data. Moreover, two statistical regression models, linear regression and gradient boosting machine, were developed and their performances were evaluated with three evaluation indices on training and test sets. Features importance is explained for both methods and the cross-validation method is used to find the best hyperparameters for the gradient boosting and linear regression algorithms. The gradient boosting machine with optimized parameters can give the best results with an R^2 value of 0.94 on the training set and 0.87 on the test set. All codes are available at the end of the report.

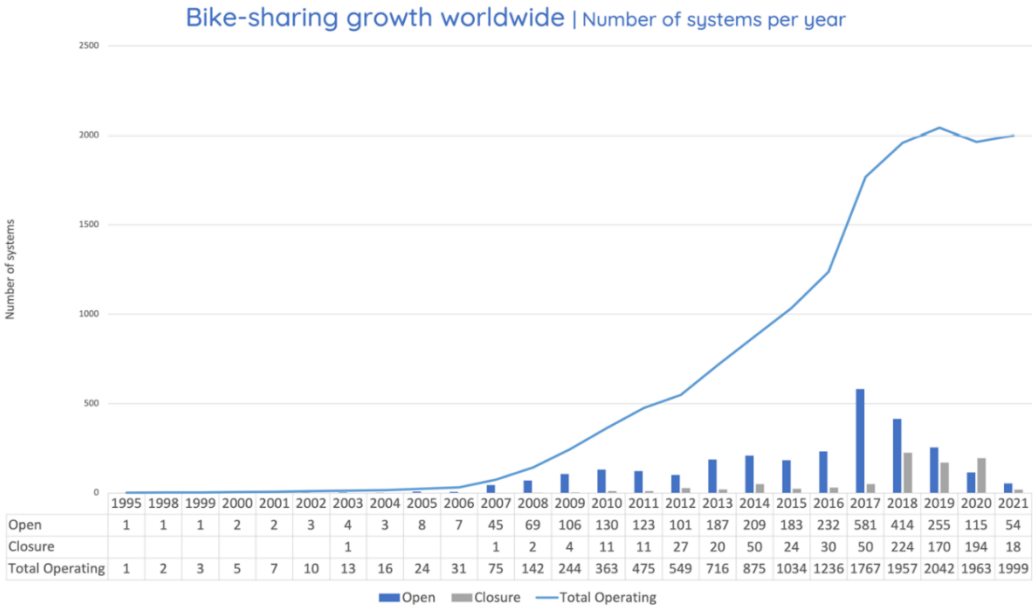
Table of Content

1. Introduction

Bike-sharing is a concept originating from the revolutionary 1960s. It was initially grow slowly, but the development in technology and bike tracking systems accelerate this progression [1]. According to the Meddin Bike-Sharing World Map, there were more than 10 million bikes shared in diverse kinds of schemes by August 2021[2]. Although our data belongs to the year between 2017 and 2018, it is still interesting to look at the recent report. Fig. 1 shows the growth of bike-sharing systems from 1995 until 2021. It can be seen that the number of systems increase dramatically in recent years. Although because of the COVID-19 pandemic, the number of system closures exceeded the number of launches in 2020, the number of openings has again grown in 2021.

A bike-sharing system provides bike rentals which are returned automatically at a network of kiosks located throughout a city. In these systems, people are able to rent a bike from one place and at the end of their trip return the bike to a different spot. Considering global attention to climate issues, introducing bike-sharing services receives increasing attention in recent decades. These services provide numerous benefits, such as promoting cycling, reducing greenhouse gas emissions, improving public health, and reducing traffic congestion.

With the growing number of users, it is important to have an accurate prediction of available bikes in the stations in order to make the system function consistently. Different studies have been done to predict bike usage using weather data. Sathishkumar et al. [3] present five statistical models for bike-sharing demand prediction based on weather data on two different data sets. In another study, Sathishkumar et al. [4] compare the performances of different algorithms in demand forecasting of Seoul bike-sharing. A short-term prediction for a case in Suzhou, China using recurrent neural networks (RNNs) and Random Forest methods is proposed by Wang et al. [5]. This report provides data analysis on a dataset containing the rented bicycles in the Seoul bike-sharing system and weather information and investigates two predictive models for demand forecasting.



	1995	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
Open	1	1	1	2	2	3	4	3	8	7	45	69	106	130	123	101	187	209	183	232	581	414	255	115	54
Closure							1				1	2	4	11	11	27	20	50	24	30	50	224	170	194	18
Total Operating	1	2	3	5	7	10	13	16	24	31	75	142	244	363	475	549	716	875	1034	1236	1767	1957	2042	1963	1999

Figure 1. Number of bike-sharing systems worldwide per year [2]

2. Methodology

2.1 Linear Regression

Linear regression is a conventional method for prediction problems. Linear regression models the relationship between a scalar response (dependent variable) and one or more explanatory variables (independent variables). It is called multiple linear regression when more than one independent variable is considered. The basic mathematical model of linear regression is assumed as in Eq. 1.

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n + \varepsilon \quad (1)$$

Here, β_0 to β_n are unknown parameters in the model, known as the regression coefficient, and ε is the term of error.

2.2 Gradient Boosting Machine

Gradient boosting machine (GBM) is one of the most powerful techniques for building predictive models for classification and regression tasks. It generates a model in the form of a collection of weak prediction models, most commonly decision trees. Gradient boosting mainly consists of three elements, a lost function to be optimized, a weak learner to make a prediction, and an additive model to add weak learners to minimize the loss function. A classic loss function, which is commonly used in practice is the mean squared error loss (Eq. 2). At each stage of gradient boosting, there is a function F_m which the algorithm tries to improve the function by adding a new estimator (h_m) and providing a new function F_{m+1} (Eq. 3).

$$Loss = \frac{1}{n}\sum(y_i - \hat{y}_i)^2 \quad (2)$$

$$F_{m+1}(x) = F_m(x) + h(x) = y \quad (3)$$

Or, equivalently

$$h(x) = y - F_m(x) \quad (4)$$

Therefore, in regression GBM with square loss, the algorithm fits h to the residuals which mean updating *F* based on negative gradient descent.

2.3 Evaluation Metrics

The evaluation metrics used in this report are Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). RMSE is a standard deviation of prediction errors or residuals. It shows how far predictions fall from observed values using Euclidean distance. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. R-squared values range from 0 to 1 and determine the proportion of variance in the dependent variable that can be explained by the independent variables. The equations for calculating RMSE, R-squared, and MAE are given in the following.

$$RSME = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n}} \quad (5)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \quad (6)$$

$$MAE = \frac{\sum_{i=1}^{n}[Y_i - \hat{Y}_i]}{n} \quad (7)$$

Here, Y_i is the actual measurement value, $\hat{Y}_i$ is the predicted value, $\bar{Y}$ is the average of the sample and n is the sample size.

3. Main Topic

3.1 Data Description

Originally, rented bike data for one year (2017 December to November 2018) comes from the Seoul Public Data Park website of South Korea, where the hourly public rental history of Seoul bikes is available[6]. The dataset used in this report was downloaded from the UCI Machine Learning Repository [7]. It contains weather information (Temperature, Humidity, Wind speed, Visibility, Dew point, Solar radiation, Snowfall, Rainfall), the number of bikes rented per hour, and date information. The dataset contains 8760 rows and 14 columns in total. Table 1 lists all features in the dataset with corresponding data type and data value space.

Table 1. Data variables and description

Parameters	**Type**	**Data Value Space**
Date	day-month-year	$S_1 = \{1/12/2017, \dots, 30/11/2018\}$
Rented Bike	Integer	$S_2 = \{0, 1, \dots, 3556\}$
Hour	Integer	$S_3 = \{0, 1, \dots, 23\}$
Temperature (℃)	Continuous	$S_4 = [-17.8, 39.4]$
Humidity (%)	Continuous	$S_5 = [0, 100]$
Wind Speed ($\frac{m}{s}$)	Continuous	$S_6 = [0, 7.4]$
Visibility ($10m$)	Continuous	$S_7 = [0, 2000]$
Dew point temperature (℃)	Continuous	$S_8 = [-30.6, 27.2]$
Solar radiation ($\frac{MJ}{m2}$)	Continuous	$S_9 = [0, 3.52]$
Rainfall (mm)	Continuous	$S_{10} = [0, 35]$
Snowfall (cm)	Continuous	$S_{11} = [0, 8.8]$
Season	Categorical	$S_{12} = \{Autumn, Winter, Spring, Summer\}$
Holiday	Categorical	$S_{13} = \{Holiday, No\ Holiday\}$
Functional Day	Categorical	$S_{14} = \{Yes, No\}$

Moreover, the dataset has been already cleaned without any missing values which mean no further steps for data cleaning are required. Table 2 describes the dataset and missing values for each feature.

The mathematical formalism of the dataset is described in the following.

- **Universe Ω:** Universe consists of Seoul city, sharing-bikes, bike-sharing system provider, and a collection of all atmospheric developments that start which is compatible with the available physical measurements
- **Elementary Events *ω*:** Each atmospheric development and any situation where someone uses a sharing bike is an observation opportunity.
- **Measurable Function (RV-function):** Procedure of measuring climate data and sharing bike usage data as well as the corresponding date.
- **The action of actually measuring Data:** Action of actually carrying out this procedure and collecting the number of bikes and weather data.
- **Data Value Space S:** The data value space would be the collection of weather information and the number of used bikes as well as the corresponding date. Data value space is achieved from Eq. (8).

$$\boldsymbol{S = S_1 \otimes \ldots \otimes S_{14}} \tag{8}$$

Table 2. A summary of the dataset and missing values

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Date                       8760 non-null   object
 1   Rented Bike Count          8760 non-null   int64
 2   Hour                       8760 non-null   int64
 3   Temperature(°C)            8760 non-null   float64
 4   Humidity(%)                8760 non-null   int64
 5   Wind speed (m/s)           8760 non-null   float64
 6   Visibility (10m)           8760 non-null   int64
 7   Dew point temperature(°C)  8760 non-null   float64
 8   Solar Radiation (MJ/m2)    8760 non-null   float64
 9   Rainfall(mm)               8760 non-null   float64
 10  Snowfall (cm)              8760 non-null   float64
 11  Seasons                    8760 non-null   object
 12  Holiday                    8760 non-null   object
 13  Functioning Day            8760 non-null   object
dtypes: float64(6), int64(4), object(4)
memory usage: 958.2+ KB
```

3.2 Data Exploration

Fig. 2 displays the number of rented bikes in the whole period. According to the figure, rental bike counts vary dramatically from hour to hour. To have a better understanding, I plotted the average count for each month in Fig. 3, and the heat map for the hourly rented bikes during the whole period from December 2017 until November 2018 in Fig. 4. From the results, it can be seen that the average demand for shared bikes is high during the summer season and less during the winter season. By looking at the heat map, we can also see a pattern related to hours of the day and the rental bikes use. This indicates that the usage increases at around 8 AM and

6 PM which can be related to the peak hours in Seoul. The histogram of rental bikes is shown in Fig. 5, and we can that there is a large tail at the right of the diagram which indicates that the data has right-skewed distribution. Fig. 6 shows the box plot for the target value with lower whisker 0, upper whisker 2400, and the median value of 505. Totally there are 147 rows that contain a rented bike count larger than 2400. It is not necessary to remove outliers from the data because there are few of them.

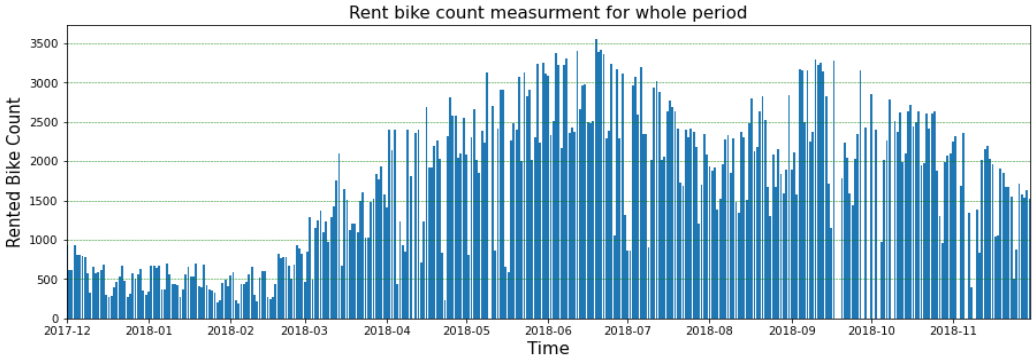


Figure 2. Hourly rented bike count in the whole period

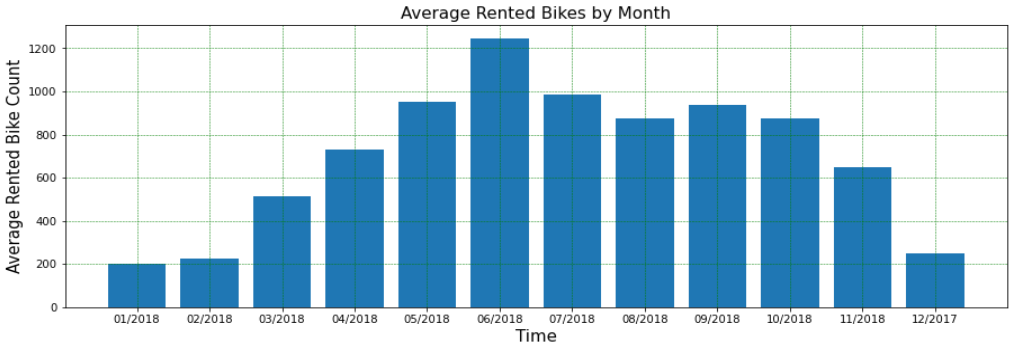


Figure 3. Average rented bikes by month

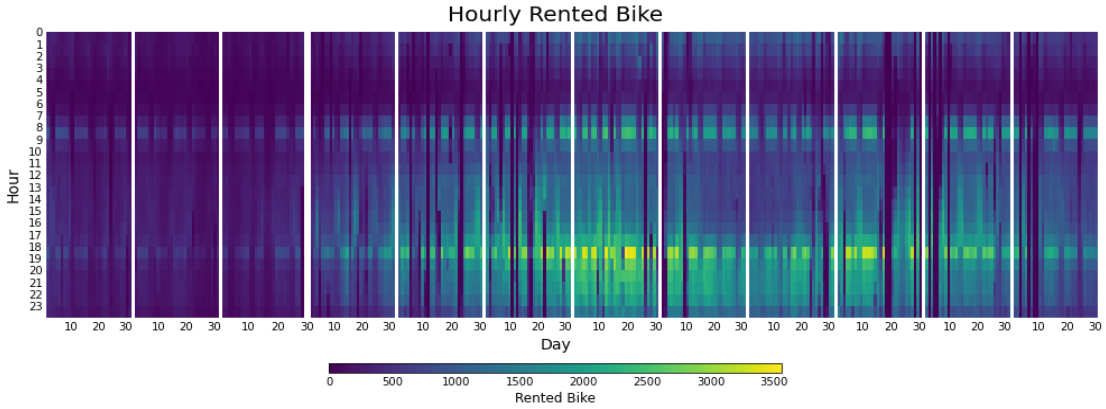


Figure 4. Heat Map for hourly rented bikes

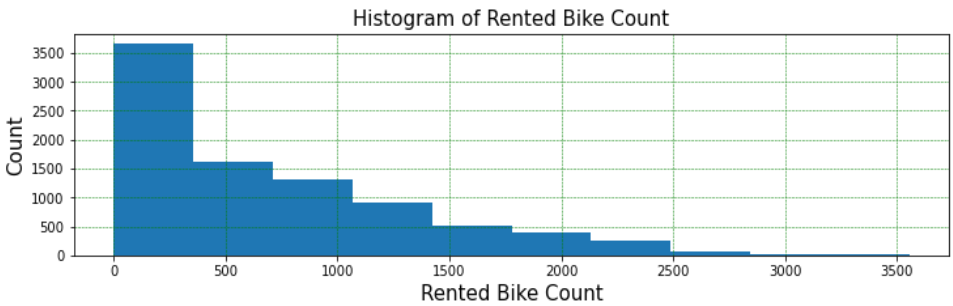


Figure 5. Histogram of rented bikes

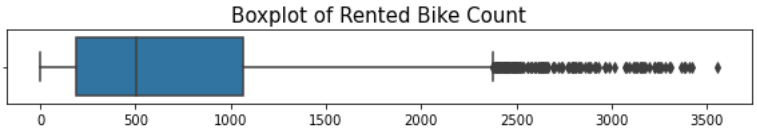


Figure 6. Boxplot of the rented bikes

Fig. 7 displays the scatter plot for each pair of numerical features in the dataset and the distribution function in the diagonal. Considering scatter plots along with Fig. 8 which presents the Pearson correlation coefficient on a heat map, we can find that the temperature feature has the highest correlation with the rented bike count (0.54). It simply means that by increasing the temperature, the demand for rented bikes will be raised. In the second position, the number of rented bikes has a positive correlation with the hour (0.41). Moreover, humidity, rainfall, and snowfall show a negative correlation with bike counts of -0.2, -0.12, and -0.14 respectively. We can conclude that these features influence the number of rented bikes negatively but not significantly. Temperature and dew point temperature features with the highest correlation coefficient of 0.91. Their linear relation can also be seen in the scatter plot. There is a negligible negative correlation of -0.036 between temperature and wind speed which means they are almost uncorrelated. I choose these two variables as an example to fit distribution functions and to find the joint pdf in the next part.

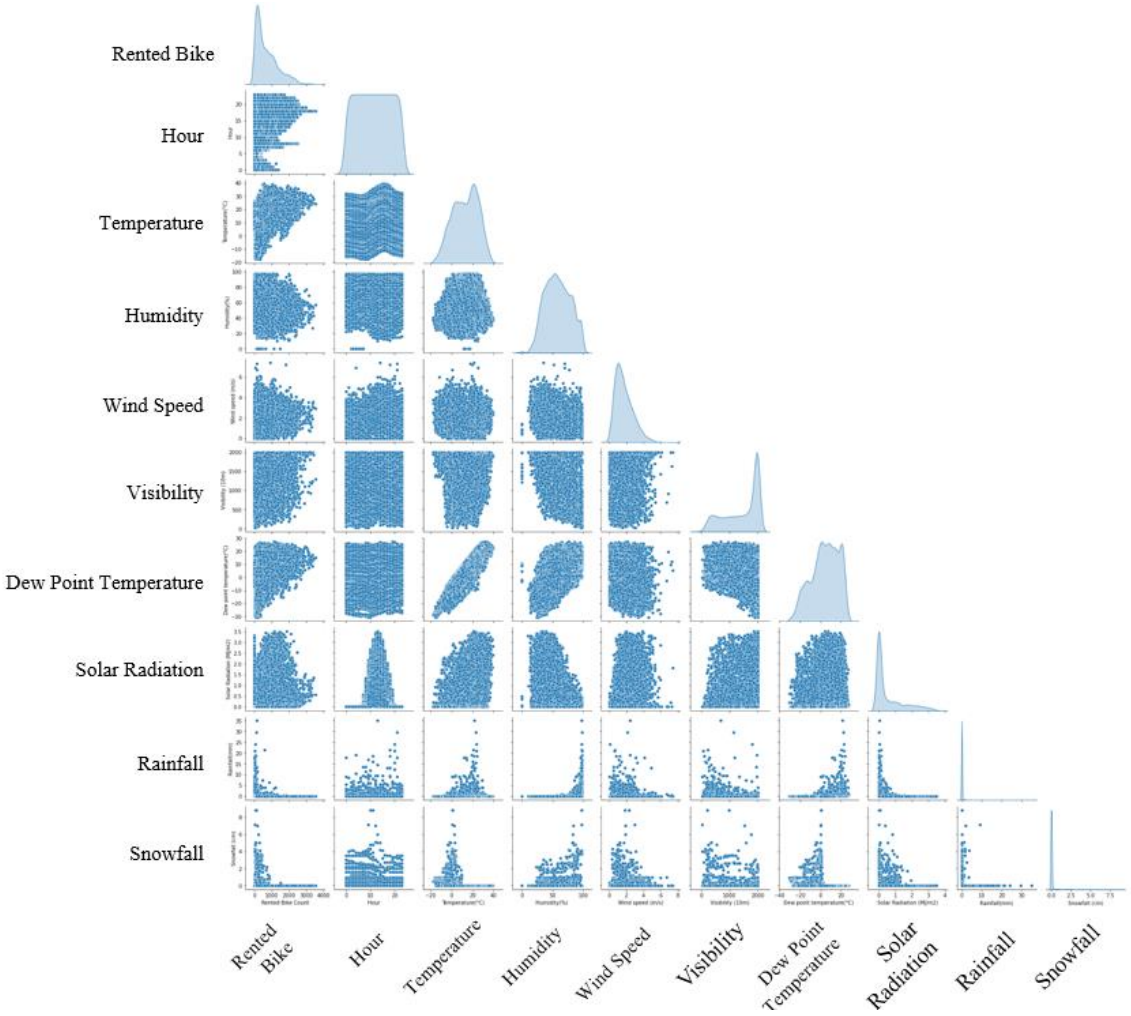


Figure 7. Scatter plots of all numerical features

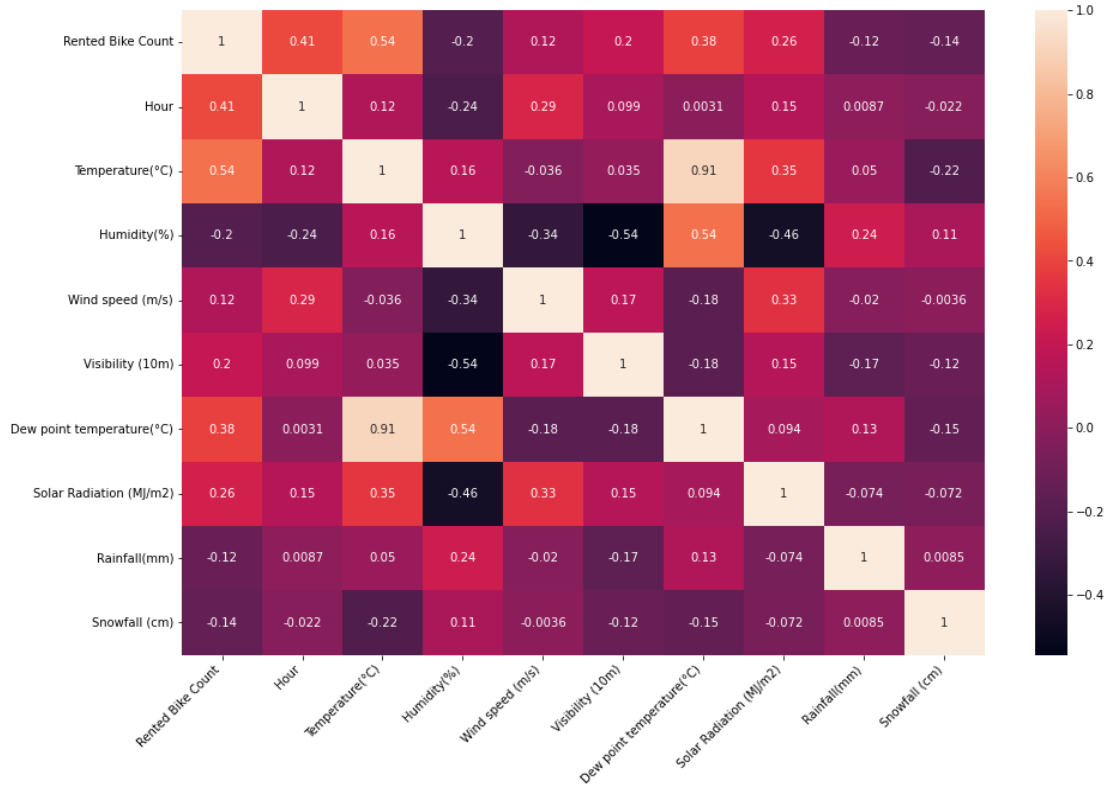


Figure 8. Correlation heatmap

3.3 Temperature and Wind Speed Data

In this part, I choose temperature and wind speed data to study their probability mass functions, probability distribution functions as well as joint probability. Fig 9 and 10 display the histogram and probability mass function for temperature and wind speed respectively. It can be seen from the histogram that the temperature data has two peaks, and it would be more accurate if we fit a multimodal normal distribution to the data. However, I chose the normal distribution with one mode to find and compare the properties such as mean and standard deviation. Moreover, the lower wind speed has a higher probability and the data is right-skewed.

In the next step, I fit different kinds of popular density functions to these features. As can be seen in Fig 11 and 12, the best estimations for temperature and wind speed density functions are normal function and gamma function respectively. The comparison between the real data and fitted distribution is given in Table 3.

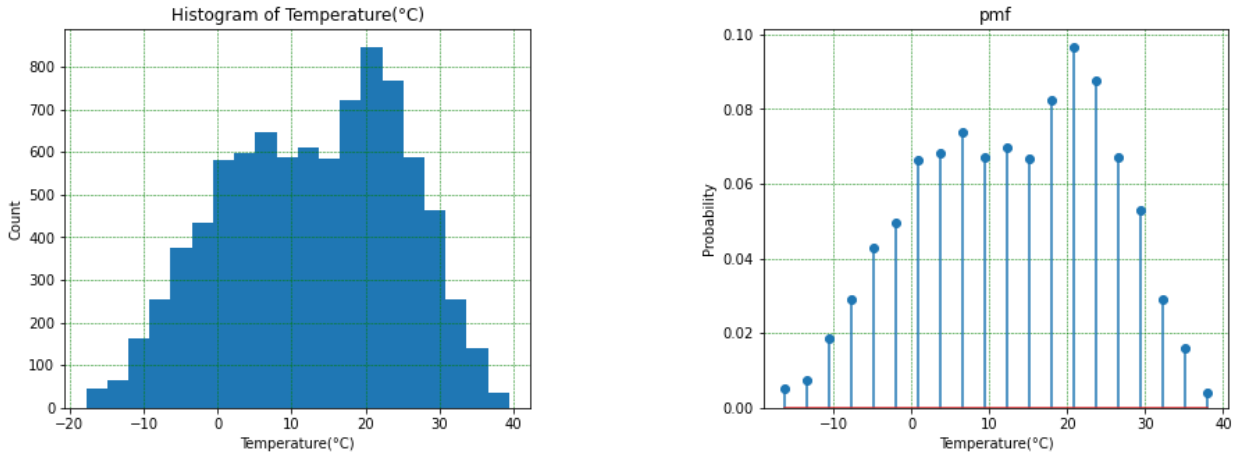


Figure 9. (a) Histogram of Temperature (b) Probability mass function of Temperature data

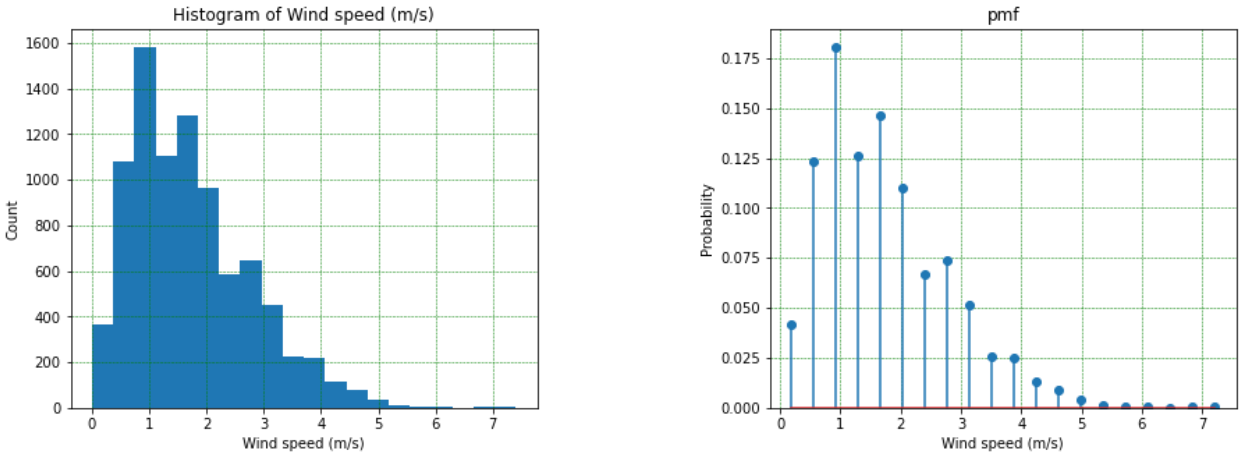


Figure 10. (a) Histogram of Wind Speed (b) Probability mass function of Wind Speed data

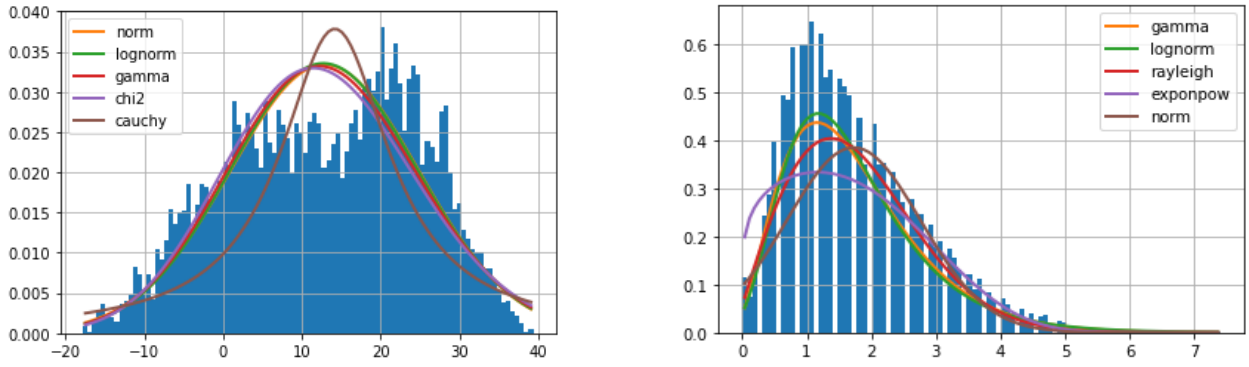


Figure 11. Fitted distribution functions to the (a) Temperature data (b) Wind Speed data

Table 3. Properties of distribution functions

Temperature	μ	σ	Skewness	Kurtosis
Normal	12.883	11.944	0	0
Real data	12.883	11.944	-0.883	-0.198

Wind Speed	μ	σ	Skewness	Kurtosis
Gamma	1.725	1.05	1.085	1.766
Real data	1.725	1.036	0.726	0.891

As an example of comparing marginal and conditional probability, Fig. 12 displays the marginal and Condit ional probability for Temperature and Temperature conditioned by Wind Speed = 3. By looking at the results, we can say that the density for Temperature conditioned by Wind Speed = 3 has almost the same pattern as Te mperature marginal density with a little difference in amounts.

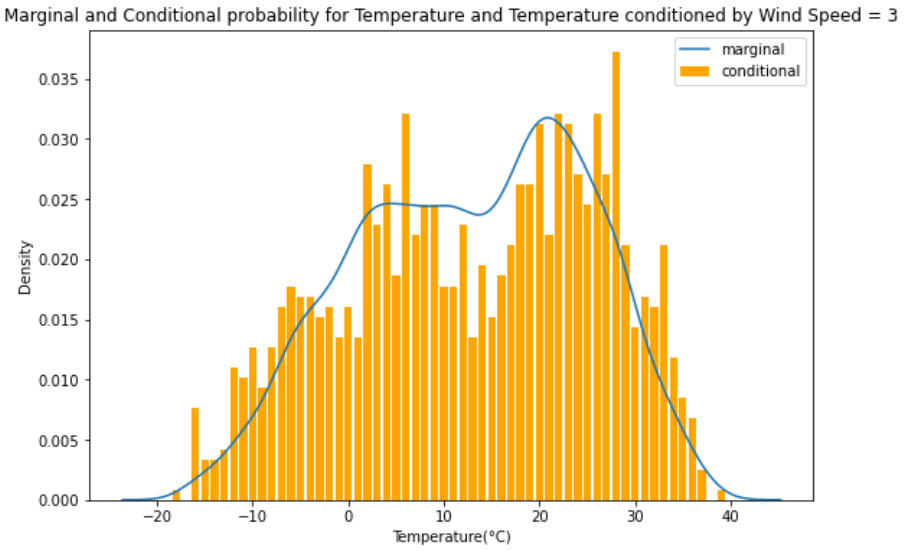


Figure 12. Marginal and Conditional probability for Temperature and Temperature conditioned by Wind Speed=3

For the next move, I round the measurements for temperature and wind speed and make discrete sets with bin sizes equal to 1 in order to find the joint probability. The calculated joint probability is shown in Fig. 13 in two different views. Moreover, the covariance matrix and means are calculated to find the multivariate Gaussian distribution (Eq. 9 and 10). The derived multivariate Gaussian distribution is plotted in Fig. 14 in the blue wireframe. We can conclude that the multivariate Gaussian could be a reasonable model for this joint probability qualitatively.

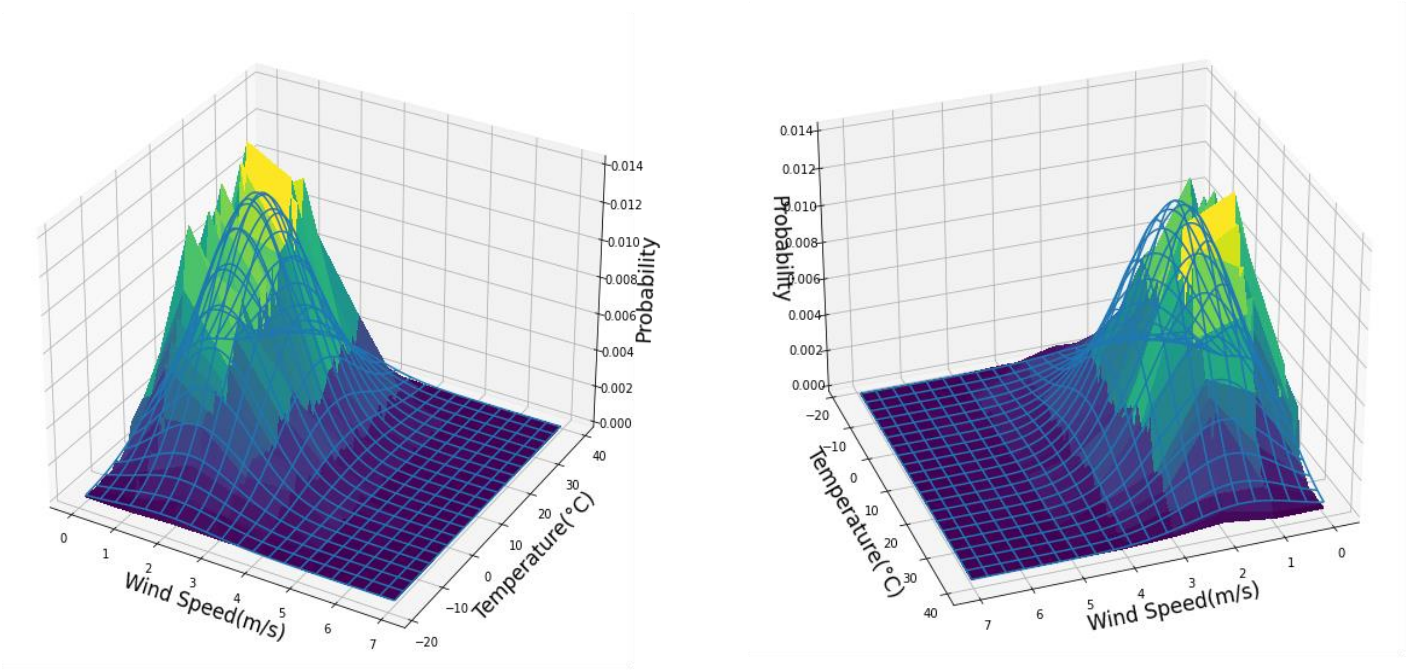


Figure 13. The joint probability of Temperature -Wind Speed and Multivariate Gaussian distribution

$$Covariance\ Matrix = \begin{bmatrix} 1.074 & -0.449 \\ -0.449 & 142.679 \end{bmatrix} \quad (9)$$

$$Means = [1.725 \quad 12.883] \quad (10)$$

3.4 Prediction

3.4.1 Data Preparation

As I mentioned before the dataset is already cleaned. In order to clearly study the effect of categorical variables on the target value, I converted the categorical variables to dummy variables. For a feature with n number of categories, it is sufficient to have n-1 dummy variables. Therefore, I dropped one column from each group of dummy variables. Moreover, the “Dew Point Temperature” feature is dropped because it is highly correlated with the “Temperature” feature in a linear way. In the next step, the dataset is divided into the training and test sets by a proportion of 75% percent for training and 25% for testing.

3.4.2 Model Development

For the first step, I developed a linear regression model to predict the target value. A grid search cross-validation was used in order to find the best number of features. The Scikit-learn library in Python provides a grid search to try all combinations of possible parameter values and find the optimal parameters for a model. From Fig. 14, we can see that the optimal number of features is 12. After that, I built the final model by choosing the optimal value.

Fig. 15 shows the residual map for the linear regression model. Residuals are the difference between the actual values and the predicted values. It can be seen that the residuals are not well distributed towards the zero line in the linear regression model. In another word, the LR model performs weakly in predicting the target values.

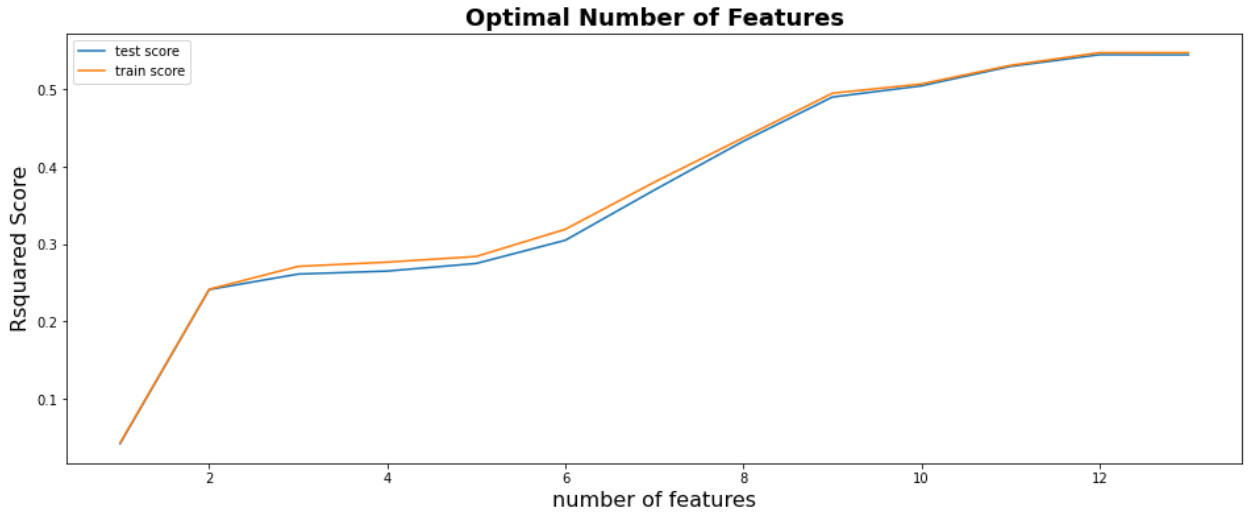


Figure 14. The R-squared score for different number of features

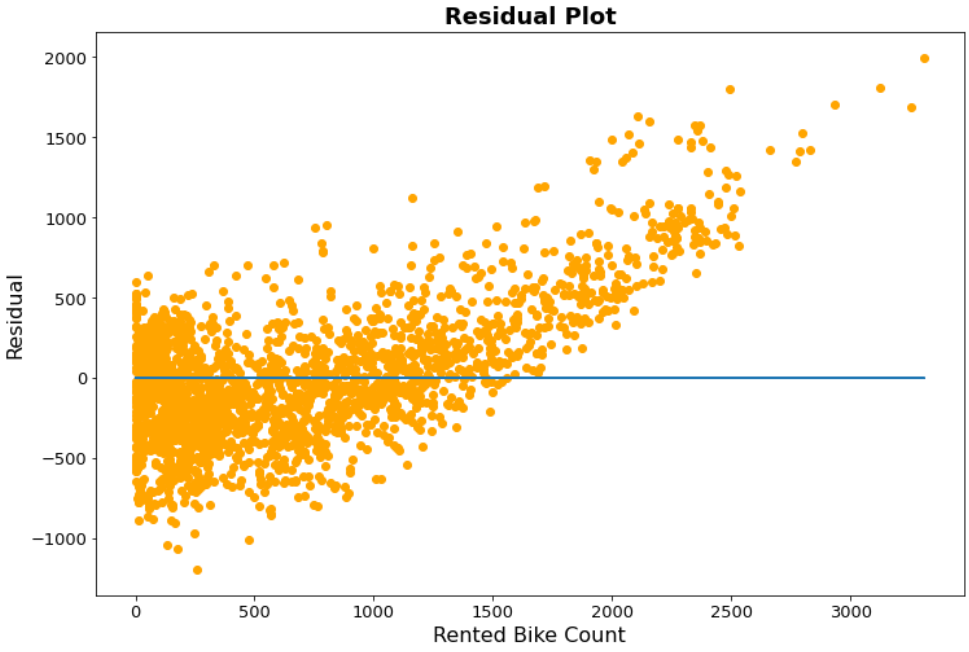


Figure 15. Residuals plot for LR

The second statistical modeling used for prediction is GBM. In order to increase the accuracy of the prediction, it is necessary to tune the hyperparameters of each regression model. Hyperparameters for Gradient Boosting Machine include the learning rate, number of estimators, and maximum depth of the tree. The grid search runs for a learning rate ranging from 0.01 to 0.001, the number of estimators ranging from 100 to 3000, and the maximum depth ranging from 2 to 14. As shown in Fig. 16 and Fig. 17, it turns out that the optimal values are 0.01, 2500, and 5 respectively.

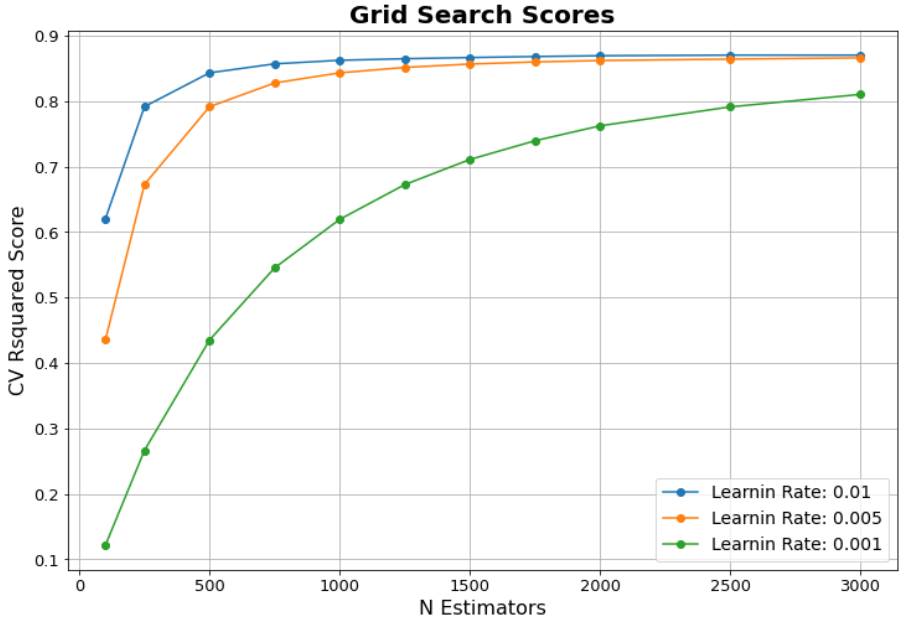


Figure 16. Grid search scores based on learning rates and the number of estimators

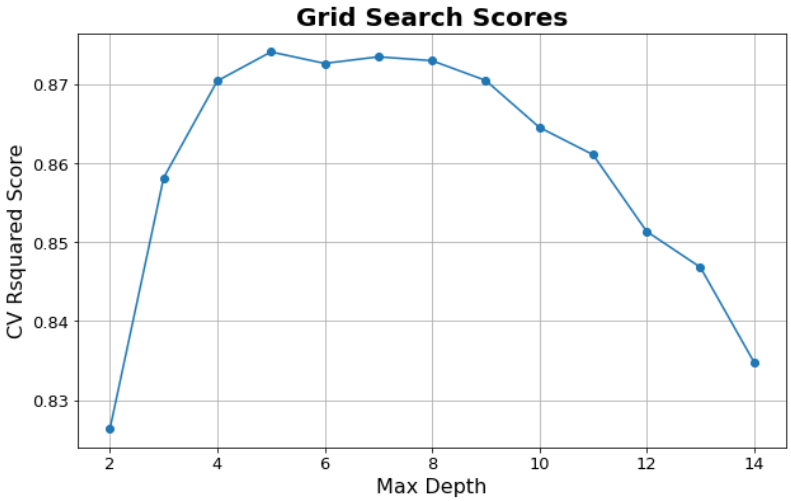


Figure 17. Grid search scores based on maximum depth

Fig. 18 and Fig. 19 exhibit the permutation importance of features in the two models. In the procedure of defining the permutation feature importance, each feature is randomly shuffled in order to break the relation between the feature and the target [8]. In this way, the drop in the model score is indicative of how much the model depends on the feature. Permutation importance does not reflect the intrinsic predictive value of a feature by itself but how important this feature is for a particular model.

The importance of each feature in the linear regression model is presented in Fig. 18. We can say that "Temperature", "Hour", and "Humidity" are the most important in the linear regression model. According to Fig. 19, the three most important features in the fitted GBM model are "Hour", "Temperature", and "Solar Radiation". In both statistical models, "Temperature", "Hour", and "Humidity" are presented as the five most predictive features.

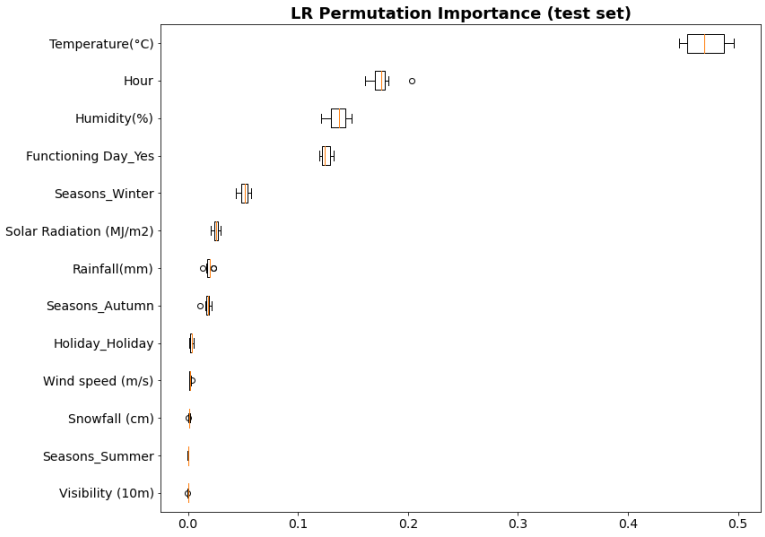


Figure 198. Feature importance for LR

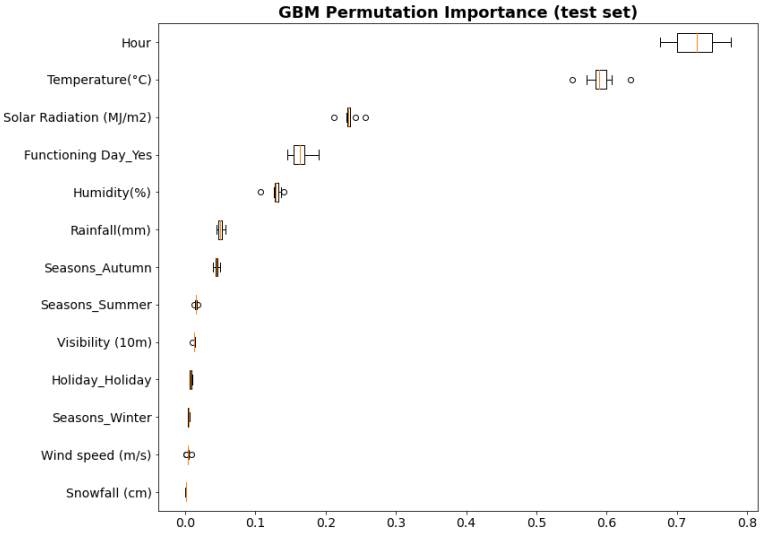


Figure 189. Feature importance for GBM

Three evaluation indices, R-squared, RMSE, and MAE are calculated for both statistical models. The best model is the model with the lowest RMSE and MAE values as we as the highest R^2 value. By looking at Table 4, we conclude that the GBM model with optimized hyperparameters is able to produce a better prediction of rented bike count. On both training and test sets, GBM has lower RMSE and MAE which means lower error, and higher R^2 which means better explanation of the fit. The R^2 value for the GBM model was 0.94 and 0.87 on the training set and testing set respectively. The linear regression model has a value of 427.70 for RMSE on the testing set, in comparison with the GBM model with a value of 145.65.

Table 4. Models Performance

Model	Training			Testing		
	R^2	RMSE	MAE	R^2	RMSE	MAE
LM	0.55	434.42	323.35	0.56	427.70	324.36
GBM	0.94	160.20	103.26	0.87	232.60	145.65

4. Conclusion and Outlook

This study focused on an exploratory data analysis of the Seoul bike-sharing service dataset along with weather and time data. The results show the relationship between weather data and rental bike users. I investigated the distribution characteristic of temperature and wind speed data which helps to have a better understanding of the features. Since the number of rented bikes changes in a short period of time, demand forecasting for the number of bikes plays a significant role in operating bike-sharing programs. I have developed two different statistical models (linear regression and gradient boosting machine) in order to predict the rented bikes. GBM model presented more accurate results with R^2, RMSE and MAE values of 0.87, 232.60, and 145.65 on the test set respectively. There are lots of possibilities of machine learning models that can be investigated in order to get better results such as support vector machines or neural networks. Moreover, using an updated dataset with more observations will help our prediction. For future work, we can focus on the district-level rental bike prediction. In this way, we can have a more accurate prediction for each district which will help services to operate in a more consistent manner.

5. Code

```
# Importing Libraries
import math
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.colors as mc
from matplotlib.cm import ScalarMappable
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import GridSearchCV
from sklearn.inspection import permutation_importance
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFE
from scipy.stats import kurtosis
from scipy.stats import skew
from fitter import Fitter, get_common_distributions, get_distributions
from scipy.stats import norm
from scipy.stats import gamma
from scipy.stats import multivariate_normal
import warnings # supress warnings
warnings.filterwarnings('ignore')

# reading csv files
data =  pd.read_csv('E:/Data-Engineering-Jacbos/Semester 2/Statistical
Modeling/Project/SeoulBikeData.csv', encoding= 'unicode_escape')
data['Date'] =  pd.to datetime(data['Date'], dayfirst= True)

# Count number of outliers
data[data['Rented Bike Count']>2400].count()

# Table 2
data.info()
```

Data Exploration

```
# Figure 2
# Plot Rented Bike Count for whole period
num_bins = 50
fig, ax = plt.subplots(figsize=(15,5))
ax.bar(data['Date'], data["Rented Bike Count"])
ax.set_title('Rent bike count measurment for whole period', fontsize = 15)
ax.set_ylabel('Rented Bike Count', fontsize = 15)
ax.set_xlabel('Time', fontsize = 15)
ax.grid(axis='y', color = 'green', linestyle = '--', linewidth = 0.5)
ax.set_xlim([min(data['Date']), max(data['Date'])])
plt.show()

# Figure 3
# Plot Average Rented Bike for each month
selected_columns = data.loc [:,("Date","Rented Bike Count")]
selected_columns['Date'] = selected_columns['Date'].dt.strftime('%m/%Y')
```

```
selected_columns_g = selected_columns.groupby ('Date').mean()
fig, ax = plt.subplots(figsize=(15,5))
ax.bar(selected_columns_g.index, selected_columns_g["Rented Bike Count"])
ax.set_title('Average Rented Bikes by Month', fontsize = 15)
ax.set_ylabel('Average Rented Bike Count', fontsize = 15)
ax.set_xlabel('Time', fontsize = 15)
ax.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()

# Figure 4
# Plot Heat map of Rented Bike Count for whole period
data1 = data
data1['Date'] =  pd.to_datetime(data1['Date'], dayfirst= True)
data1 = data1.set_index('Date')
data1.index += pd.TimedeltaIndex(data1['Hour'], unit='h')
MIN_WS = data1['Rented Bike Count'].min() #edit!!
MAX_WS = data1['Rented Bike Count'].max() #edit!!

# Plotting heatmap for whole period
def single_plot (data, month, ax):
    data = data[data.index.month == month]
    hour = data.index.hour
    day = data.index.day
    WindSpeed = data['Rented Bike Count']
    #WindSpeed = WindSpeed.values.reshape(24, len(day.unique()), order="F")
    d = {'hour':hour, 'day':day, 'windspeed':WindSpeed}
    df = pd.DataFrame (data=d)
    WindSpeed = df.pivot_table(index="hour",columns="day",values="windspeed", aggfunc="mean")
    xgrid = np.arange(day.max() + 1) + 1
    ygrid = np.arange(25)
    ax.pcolormesh(xgrid, ygrid, WindSpeed, cmap="viridis", vmin=MIN_WS, vmax=MAX_WS)
    # Invert the vertical axis
    ax.set_ylim(24, 0)
    # Set tick positions for both axes
    ax.yaxis.set_ticks([i for i in range(24)])
    ax.xaxis.set_ticks([10, 20, 30])
    # Remove ticks by setting their length to 0
    ax.yaxis.set_tick_params(length=0)
    ax.xaxis.set_tick_params(length=0)

    # Remove all spines
    ax.set_frame_on(False)


fig, axes = plt.subplots(1, 12, figsize=(14, 5), sharey=True)
single_plot(data1, 12, axes[0])
for i in range(1,12):
    single_plot(data1, i, axes[i])

fig.subplots_adjust(left=0.05, right=0.98, top=0.9, hspace=0.08, wspace=0.04)

# Make some room for the legend in the bottom.
fig.subplots_adjust(bottom=0.15)

# Create a new axis to contain the color bar
cbar_ax = fig.add_axes([0.3, 0.005, 0.4, 0.025])

# Create a normalizer that goes from minimum to maximum wind speed
norm = mc.Normalize(MIN_WS, MAX_WS)
```

```
# Create the colorbar and set it to horizontal
cb = fig.colorbar(
    ScalarMappable(norm=norm, cmap="viridis"),
    cax=cbar_ax, # Pass the new axis
    orientation = "horizontal")

# Remove tick marks
cb.ax.xaxis.set_tick_params(size=0)

# Set legend label
cb.set_label("Rented Bike", size=12)

# Set common labels for x and y axes
fig.text(0.5, 0.08, "Day", ha="center", va="center", fontsize=14)
fig.text(0.02, 0.5, 'Hour', ha="center", va="center", rotation="vertical", fontsize=14)
fig.suptitle("Hourly Rented Bike", fontsize=20, y=0.97)
fig

# Figure 5
# Plot histogram for Rented Bike Count
fig, ax = plt.subplots(figsize=(12,3))
ax.hist(data ["Rented Bike Count"])
ax.set_title('Histogram of Rented Bike Count',fontsize = 15)
ax.set_xlabel('Rented Bike Count', fontsize = 15)
ax.set_ylabel('Count', fontsize = 15)
ax.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.show()

# Figure 6
# Plot Boxplot for Rented Bike Count
fig, ax = plt.subplots(figsize=(10,1))
sns.boxplot(data=data,x=data["Rented Bike Count"])
plt.title("Boxplot of Rented Bike Count", fontsize = 15)
ax.set_xlabel('')
plt.show()

# Figure 7
# Scatter plot for all features
sns.pairplot(data, diag_kind='kde', corner=True)
plt.suptitle('Scatter Plot for all features', y = 1, fontsize = 25)
plt.show()

# Figure 8
# Correlation Matrix heatmap based on Pearson method
corrMatrix = data.corr(method='pearson')
fig, ax = plt.subplots(figsize=(15,10))
corrChart = sns.heatmap(corrMatrix, annot=True)
corrChart.set_xticklabels(corrChart.get_xticklabels(), rotation=45, horizontalalignment='right')
plt.show()
```

Temperature Data

```
# Figure 9
# Plot histogram and pmf for Temperature data
num_bins = 20
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(15,5))
ax1.hist(data ["Temperature(°C)"],bins=num_bins)
ax1.set_title('Histogram of Temperature(°C)')
ax1.set_xlabel('Temperature(°C)')
ax1.set_ylabel('Count')
```

```
ax1.grid(color = 'green', linestyle = '--', linewidth = 0.5)

counts , bins = np.histogram(data ["Temperature(°C)"],bins=num_bins)
bins = bins[:-1] + (bins[1] - bins[0])/2
probs = counts/float(counts.sum())
#print (probs.sum()) # 1.0
ax2.stem(bins, probs)
ax2.set_title('pmf')
ax2.set_xlabel('Temperature(°C)')
ax2.set_ylabel('Probability')
ax2.set_ylim(bottom=0)
ax2.grid(color = 'green', linestyle = '--', linewidth = 0.5)

plt.subplots_adjust(wspace=0.5)
plt.show()

# Table 3
# Calculate distribution properties for Temperature data
kur = kurtosis(data['Temperature(°C)'])
skw = skew(data['Temperature(°C)'])
std = np.std(data['Temperature(°C)'])
mean = np.mean(data['Temperature(°C)'])
print ('Mean =',round(mean,3))
print ('Standard deviation =',round(std,3))
print ('Kurtosis =', round(kur,3))
print ('Skewness =',round(skw,3))

# Figure 11.a
# Fit distribution functions to the Temperature data
f = Fitter(data['Temperature(°C)'],distributions= get_common_distributions(),timeout=60)
f.fit()
f.summary()

# Table 3
# Caculate distribution properties of Normal distribution
mean, var, skw, kurt = norm.stats(f.fitted_param['norm'][0],f.fitted_param['norm'][1],moments='mvsk')
print('Mean =',np.around(mean, decimals = 3))
print('Standard deviation =', np.around(math.sqrt(var), decimals = 3))
print('Skewness =', skw)
print ('Kurtosis =', kurt)
```

Wind Speed Data

```
# Figure 10
# Plot histogram and pmf for Temperature data
num_bins = 20
fig, (ax1, ax2) = plt.subplots(1, 2,figsize=(15,5))

ax1.hist(data ["Wind speed (m/s)"],bins=num_bins)
ax1.set_title('Histogram of Wind speed (m/s)')
ax1.set_xlabel('Wind speed (m/s)')
ax1.set_ylabel('Count')
ax1.grid(color = 'green', linestyle = '--', linewidth = 0.5)

counts , bins = np.histogram(data ["Wind speed (m/s)"],bins=num_bins)
bins = bins[:-1] + (bins[1] - bins[0])/2
probs = counts/float(counts.sum())

ax2.stem(bins, probs)
ax2.set_title('pmf')
```

```
ax2.set_xlabel('Wind speed (m/s)')
ax2.set_ylabel('Probability')
ax2.set_ylim(bottom=0)
ax2.grid(color = 'green', linestyle = '--', linewidth = 0.5)
plt.subplots_adjust(wspace=0.5)
plt.show()

# Table 3
# Calculate distribution properties for Wind Speed data
kur = kurtosis(data['Wind speed (m/s)'])
skw = skew(data['Wind speed (m/s)'])
std = np.std(data['Wind speed (m/s)'])
mean = np.mean(data['Wind speed (m/s)'])
print ('Mean =',round(mean,3))
print ('Standard deviation =',round(std,3))
print ('Kurtosis =', round(kur,3))
print ('Skewness =',round(skw,3))

# Figure 11.b
# Fit distribution functions to the Wind Speed data
f = Fitter(data['Wind speed (m/s)'],distributions= get_common_distributions(),timeout=60)
f.fit()
f.summary()

# Table 3
# Caculate distribution properties of Gamma distribution
mean, var, skw, kurt =
gamma.stats(f.fitted_param['gamma'][0],f.fitted_param['gamma'][1],f.fitted_param['gamma'][2]
                                    , moments='mvsk')
print('Mean =', np.around(mean, decimals = 3))
print('Standard deviation =', np.around(math.sqrt(var), decimals = 3))
print('Skewness =', np.around(skw, decimals = 3))
print ('Kurtosis =', np.around(kurt, decimals = 3))
```

Joint Probability

```
# Calculate joint pmf for Temperature and Wind Speed
    # Round the temperature and wind speed to map on the a discrete S with interval width d=1
t = np.array(round(data['Temperature(°C)']), dtype=object)
w = np.array(round(data['Wind speed (m/s)']), dtype=object)

# Build crosstab to find the frequency of each temperature and wind speed measurement
cnt = pd.crosstab(t, w, rownames=['Temperature(°C)'], colnames=['Wind speed (m/s)'])
# Calculate probability based on frequency
cnt = cnt / cnt.values.sum()

# Equation 9
# Caculate Covariance Matrix
m = np.array([data['Wind speed (m/s)'],data['Temperature(°C)']])
arr_cov = np.cov(m)
print ('Covariance Matrix: \n', np.around(arr_cov,3))

# Equation 10
# Calculate mean
arr_mean = np.array([data['Wind speed (m/s)'].mean(),data['Temperature(°C)'].mean()])
print ('Means Array: \n', arr_mean)
print ('\nMean for Wind Speed =', round(arr_mean[0],3))
print ('Means for Temperature =', round(arr_mean[1],3))
```

```
# Figure 13
# Generating a Gaussian bivariate distribution with given mean and covariance matrix
random_seed=1000
mean = arr_mean
cov = arr_cov
distr = multivariate_normal(cov = cov, mean = mean, seed = random_seed)

# Generating a meshgrid complacent with
mean_1, mean_2 = mean[0], mean[1]
sigma_1, sigma_2 = cov[0,0], cov[1,1]
x = np.linspace(0,7,200)
y = np.linspace(-18,39,200)
X, Y = np.meshgrid(x,y)
x1 = np.linspace(0,7,8)
y1 = np.linspace(-18,39,58)
X1, Y1 = np.meshgrid(x1,y1)

    # Generating the density function
    # for each point in the meshgrid
pdf = np.zeros(X.shape)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        pdf[i,j] = distr.pdf([X[i,j], Y[i,j]])

fig, ax = plt.subplots(figsize=(15,10),subplot_kw={"projection": "3d"})

ax.plot_surface(X1, Y1, cnt,cmap='viridis',linewidth=0,antialiased=False)
ax.plot_wireframe(X, Y, pdf, rstride=10, cstride=10)
ax.set_xlabel('Wind Speed(m/s)', fontsize = 17)
ax.set_ylabel('Temperature(°C)', fontsize = 17)
ax.set_zlabel('Probability', fontsize = 17)
plt.show()
```

Marginal and conditional Probability

```
# Calculate marginal distributions
marginal_wind = []
marginal_temp = []
for i in range(0,8):
    marginal_wind.append(cnt[i].sum())
for i in range(-18,40):
    marginal_temp.append(cnt.loc[i,:].sum())
cnt_resin = cnt.reset_index(drop=True)

# Figure 12
# Calculate conditional probability for temperature conditioned by wind speed = 3
con_t_w3 = cnt_resin.loc[:,3]/marginal_wind[3]

# Plotting marginal and conditional density
    # temperature marginal compared with temperature conditioned by wind speed = 3
T = np.linspace(-18,39,58)
plt.figure(figsize=(9, 6))
sns.kdeplot(data['Temperature(°C)'])
#plt.stem(T, con_t_w3, linefmt='orange', markerfmt='orange', basefmt='orange')
plt.bar(T,con_t_w3, color='orange')
plt.legend(['marginal', 'conditional'])
plt.title('Marginal and Conditional probability for Temperature and Temperature conditioned by Wind
Speed = 3')
```

Data preparation

```
# Divide dataset into dependent and independent variables
X = data.drop (columns = ['Date' , 'Rented Bike Count'])
Y = data['Rented Bike Count']

# Convert categorical variables into dummy variables
X = pd.get_dummies(X)
# n-1 variables sufficient for each categorical feature
X = X.drop (columns = ['Dew point temperature(°C)','Seasons_Spring','Holiday_No Holiday','Functioning
Day_No'])

# Split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.25, random_state=13)
```

Linear Regression

```
# step-1: create a cross-validation scheme
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

# step-2: specify range of hyperparameters to tune
hyper_params = [{'n_features_to_select': list(range(1, 14))}]

# step-3: perform grid search
# 3.1 specify model
lm = LinearRegression()
lm.fit(x_train, y_train)
rfe = RFE(lm)

# 3.2 call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe, param_grid = hyper_params,scoring= 'r2',
                        cv = folds, verbose = 1,return_train_score=True)

# fit the model
model_cv.fit(x_train, y_train)
cv_results = pd.DataFrame(model_cv.cv_results_)

# Figure 14
# Plot cv results
plt.figure(figsize=(16,6))
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plt.xlabel('number of features',fontsize=16)
plt.ylabel('Rsquared Score',fontsize=16)
plt.title("Optimal Number of Features", fontsize=18, fontweight ='bold')
plt.legend(['test score', 'train score'], loc='upper left')

# final model
n_features_optimal = 12
lm = LinearRegression()
lm.fit(x_train, y_train)
rfe = RFE(lm, n_features_to_select=n_features_optimal)
rfe = rfe.fit(x_train, y_train)

# Table 4
# Calculate evaluation metrics for LR on test set

rmse_lr = np.sqrt(mean_squared_error(y_test, lm.predict(x_test)))
```

```
y pred =  lm.predict(x test)

print("The Rsquared on test set: {:.4f}".format(lm.score(x_test, y_test)))
print("The Root Mean Squared Error (RMSE) on test set: {:.4f}".format(rmse_lr))
print("The Mean Absolute Error (MAE) on test set: {:.4f}".format(mean_absolute_error(y_test, y_pred)))

# Table 4
# Calculate evaluation metrics for LR on training set
rmse_lr = np.sqrt(mean_squared_error(y_train, lm.predict(x_train)))
y_pred =  lm.predict(x_train)
print("The Rsquared on train set: {:.4f}".format(lm.score(x_train, y_train)))
print("The Root Mean Squared Error (RMSE) on train set: {:.4f}".format(rmse_lr))
print("The Mean Absolute Error (MAE) on train set: {:.4f}".format(mean_absolute_error(y_train,
y_pred)))

# Figure 15
# Plot Residuals
y_pred =  lm.predict(x_test)
residuals = y_test - y_pred
fig = plt.figure(figsize=(12, 8))
plt.plot(y_test,residuals, 'o', color='orange')
plt.plot(y_test, [0]*len(y_test))
plt.title("Residual Plot", fontsize=18, fontweight ='bold')
plt.xlabel("Rented Bike Count", fontsize=16)
plt.ylabel("Residual", fontsize=16)
plt.yticks(fontsize=13)
plt.xticks(fontsize=13)
plt.show()

# Figure 18
# Plot Permutation importance for LR
result = permutation_importance(lm, x_test, y_test, n_repeats=10, random_state=42, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
fig = plt.figure(figsize=(12, 10))
plt.boxplot(result.importances[sorted_idx].T, vert=False, labels=np.array(X.columns)[sorted_idx])
plt.title("LR Permutation Importance (test set)", fontsize=18, fontweight ='bold')
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.show()
```

Gradient Boosting Machine

```
# Search for the best value of learning_rate and n_estimators parameters
p_test1 = {'learning_rate':[0.01,0.005,0.001],
           'n_estimators':[100,250,500,750,1000,1250,1500,1750,2000,2500,3000]}

tuning1 = GridSearchCV(estimator =GradientBoostingRegressor(
    max_depth=4, min_samples_split=2, loss="squared_error", random_state=10),
            param_grid = p_test1, scoring='r2',n_jobs=4, cv=5)

tuning1.fit(x_train,y_train)
tuning1.best_params_

# Figure 16
# Define a function to plot the results of Grid Search
def plot_grid_search(cv_results, grid_param_1, grid_param_2, name_param_1, name_param_2):
    # Get Test Scores Mean and std for each grid search
    scores_mean = cv_results['mean_test_score']
    scores_mean = np.array(scores_mean).reshape(len(grid_param_2),len(grid_param_1))
    scores_sd = cv_results['std_test_score']
```

```
    scores_sd = np.array(scores_sd).reshape(len(grid_param_2),len(grid_param_1))

    # Plot Grid search scores
    _, ax = plt.subplots(1,1,figsize=(12, 8))

    # Param1 is the X-axis, Param 2 is represented as a different curve (color line)
    for idx, val in enumerate(grid_param_2):
        ax.plot(grid_param_1, scores_mean[idx,:], '-o', label= name_param_2 + ': ' + str(val))

    ax.set_title("Grid Search Scores", fontsize=20, fontweight='bold')
    ax.set_xlabel(name_param_1, fontsize=16)
    ax.set_ylabel('CV Rsquared Score', fontsize=16)
    ax.legend(loc="best", fontsize=14)
    plt.yticks(fontsize=13)
    plt.xticks(fontsize=13)
    ax.grid('on')

# Calling Method
plot_grid_search(tuning1.cv_results_, p_test1['n_estimators'], p_test1['learning_rate'], 'N
Estimators', 'Learnin Rate')

p_test2 = {'max_depth':[2,3,4,5,6,7,8,9,10,11,12,13,14]}

tuning2 = GridSearchCV(estimator =GradientBoostingRegressor(
    learning_rate=0.01,n_estimators=2500, min_samples_split=2,loss="squared_error", random_state=10),
            param_grid = p_test2, scoring='r2',n_jobs=4, cv=5)
tuning2.fit(x_train,y_train)
tuning2.best_params_

# Figure 17
name_param_1 = 'Max Depth'
cv_results = tuning2.cv_results_

# Get Test Scores Mean and std for each grid search
scores_mean = cv_results['mean_test_score']
#scores_mean = np.array(scores_mean).reshape(len(grid_param_2),len(grid_param_1))

scores_sd = cv_results['std_test_score']
#scores_sd = np.array(scores_sd).reshape(len(grid_param_2),len(grid_param_1))

# Plot Grid search scores
_, ax = plt.subplots(1,1,figsize=(10, 6))

# Param1 is the X-axis, Param 2 is represented as a different curve (color line)
#for idx, val in enumerate(grid_param_2):
#    ax.plot(grid_param_1, scores_mean[idx,:], '-o', label= name_param_2 + ': ' + str(val))
ax.plot(p_test2['max_depth'], scores_mean, '-o')

ax.set_title("Grid Search Scores", fontsize=20, fontweight='bold')
ax.set_xlabel(name_param_1, fontsize=16)
ax.set_ylabel('CV Rsquared Score', fontsize=16)
plt.yticks(fontsize=13)
plt.xticks(fontsize=13)
ax.grid('on')

# Fit a GBM model with optimized parameters
params = {"n_estimators": 2500,"max_depth": 5,"min_samples_split": 2,"learning_rate": 0.01,
    "loss": "squared_error", "random_state": 10}

reg = GradientBoostingRegressor(**params)
```

```
reg.fit(x_train, y_train)

# Table 4
# Calculate evaluation metrics on test set
rmse = np.sqrt(mean_squared_error(y_test, reg.predict(x_test)))
y_pred =  reg.predict(x_test)
print("The Rsquared on test set: {:.4f}".format(reg.score(x_test, y_test)))
print("The Root Mean Squared Error (RMSE) on test set: {:.4f}".format(rmse))
print("The Mean Absolute Error (MAE) on test set: {:.4f}".format(mean_absolute_error(y_test, y_pred)))

# Table 4
# Calculate evaluation metrics on training set
rmse = np.sqrt(mean_squared_error(y_train, reg.predict(x_train)))
y_pred =  reg.predict(x_train)

print("The Rsquared on training set: {:.4f}".format(reg.score(x_train, y_train)))
print("The Root Mean Squared Error (RMSE) on training set: {:.4f}".format(rmse))
print("The Mean Absolute Error (MAE) on training set: {:.4f}".format(mean_absolute_error(y_train,
y_pred)))

# Figure 19
# Plot Permutation importance for GBM
result = permutation_importance(
    reg, x_test, y_test, n_repeats=10, random_state=42, n_jobs=2)
sorted_idx = result.importances_mean.argsort()
fig = plt.figure(figsize=(12, 10))
plt.boxplot(result.importances[sorted_idx].T,vert=False, labels=np.array(X.columns)[sorted_idx])
plt.title("GBM Permutation Importance (test set)", fontsize=18, fontweight ='bold')
plt.yticks(fontsize=14)
plt.xticks(fontsize=14)
plt.show()
```

6. References

[1] P. DeMaio, “Bike-sharing: History, Impacts, Models of Provision, and Future,” *Journal of Public Transportation*, vol. 12, no. 4, 2009, doi: 10.5038/2375-0901.12.4.3.

[2] R. Meddin and P. J. DeMaio, “The Meddin Bike-sharing World Map,” 2020.

[3] S. v E and Y. Cho, “A rule-based model for Seoul Bike sharing demand prediction using weather data,” *European Journal of Remote Sensing*, vol. 53, no. sup1, 2020, doi: 10.1080/22797254.2020.1725789.

[4] S. v. E, J. Park, and Y. Cho, “Using data mining techniques for bike sharing demand prediction in metropolitan city,” *Computer Communications*, vol. 153, 2020, doi: 10.1016/j.comcom.2020.02.007.

[5] B. Wang and I. Kim, “Short-term prediction for bike-sharing service using machine learning,” in *Transportation Research Procedia*, 2018, vol. 34. doi: 10.1016/j.trpro.2018.11.029.

[6] “Public data for all Seoul citizens,” 2018. https://data.seoul.go.kr/index.do

[7] “UCI Machine Learning Repository.” https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand

[8] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, 2001, doi: 10.1023/A:1010933404324.